Aprendizagem Supervisionada em Machine Learning (Regressão)
Previsão da diferença de golos em jogos de futebol

Trabalho realizado por:

- Bernardo Costa Moreira - up201604014
- Filipe Carlos de Almeida Duarte da Cunha Nogueira - up201604129
- Francisco Jorge de Almeida Henriques Pereira - up201605306

Introdução

Este trabalho tem como finalidade aplicar os conceitos dados nas aulas teóricas e práticas sobre a aprendizagem supervisionada, mais especificamente regressão, sobre os dados fornecidos pelo Kaggle (European Soccer Database). No fim, os algoritmos usados deveriam fornecer uma previsão da diferença de golos resultante de cada jogo.

European Soccer Database fornecida pelo Kaggle é uma base de dados que contém toda a informação sobre os jogadores, equipas, jogos, etc, utilizada neste trabalho. Estes dados foram gerados a partir de situações reais e são recolhidos anualmente.

Devido ao grande volume de dados, esta tarefa torna-se perfeita para a aplicação de técnicas de Machine Learning. Os algoritmos de Machine Learning, a partir de entradas amostrais, funcionam através da construção de um modelo que é utilizado para fazer previsões ou decisões de acordo com os dados fornecidos. Deste modo, neste trabalho iremos utilizar algoritmos de aprendizagem supervisionada, nomeadamente Árvores de Decisão, Redes Neuronais e Suporte de Máquinas Vetoriais, para resolver o problema do tipo regressão mencionado anteriormente.

Descrição da Base de Dados e do Problema

A base de dados fornecida contém informação sobre jogos de futebol ao longo do período de 2008 a 2016. Está incluído conteúdo respetivo a mais de 11 ligas e respectivas equipas, dados sobre mais de 10000 jogadores e mais de 25000 partidas. É possível ainda encontrar informação de odds relativo a cada partida, nomeadamente odds da vitória da equipa de casa, empate ou vitória da equipa de fora.

É possível encontrar no site onde a base de dados é disponibilizada (Kaggle), um aviso quanto à falta de informação associada a algumas partidas, aspeto a ter em conta posteriormente no pré-processamento de dados.

A informação está distribuída por várias tabelas, sendo estas:
*Nota: é possível notar nos atributos de alguma tabelas a terminação _api_id que contém “foreign keys” das fontes originais de dados.

- Country
  - id (identificador do país)
  - name (nome do país)
- League:
  - id (identificador da liga)
  - country_id (relativo ao país em que a liga está inserida - Tabela Country)
  - name (nome da liga)
- Match
  - id (identificador match)
  - country_id (relativo ao país em que a liga está inserida - Tabela Country)
  - league_id (relativo à liga em que está inserida - Tabela Country)
  - season (temporada)
  - stage (jornada)
  - date (data do jogo)
  - match_api_id (identificador da partida)
  - home_team_api_id (identificador da equipa da casa)
  - away_team_api_id (identificador da equipa de fora)
  - home_team_goal (golos da equipa da casa)
  - away_team_goal (golos da equipa de fora)

    - (home/away)*player*(Xn/Yn) (posições dos jogadores no alinhamento)
    - (home/away)*player*(n) (identificador dos jogadores que jogaram a partida - player_api_id)
    - goal/shot_on/shot_off/foul_commit/card/cross/corner/possession (detalhes após jogo)
    - (gambling_house_name)H (odds da vitória para a equipa da casa)
    - (gambling_house_name)D (odds para empate)
    - (gambling_house_name)A (odds da vitória para a equipa de fora)
- Player
    - id (identificador do jogador nesta tabela)
    - player_api_id (identificador do jogador)
    - player_name (nome do jogador)
    - player_fifa_api_id (identificador do jogador nos dados do FIFA)
    - birthday (aniversário)
    - height (altura)
    - weight (peso)
- Player_Attributes
    - id (identificador do jogador nesta tabela)
    - player_fifa_api_id (identificador do jogador nos dados do FIFA)
    - player_api_id (identificador do jogador)
    - date (data de registo das informações do tuplo ao qual este atributo pertence)
    - overall_rating (rating geral do jogador)
    - potential (potêncial)
    - preferred_foot (pé de preferência do jogador)
    - attacking_work_rate (taxa de ofensiva do jogador)
    - defensive_work_rate (taxa defensiva do jogador)
    - crossing (habilidade de cruzamento)
- Team
    - id (identificador da equipa nesta tabela)
    - team_api_id (identificador da equipa)
    - team_fifa_api_id (identificador da equipa do FIFA)
    - team_long_name (nome completo da equipa)
    - team_short_name (abreviatura do nome da equipa)
- Team_Attributes
    - id (identificador da equipa nesta tabela)
    - team_fifa_api_id (identificador da equipa do FIFA)
    - team_api_id (identificador da equipa)
    - date (data de registo das informações do tuplo ao qual este atributo pertence)
    - buildUpPlaySpeed (velocidade de construção de jogadas da equipa)
    - buildUpPlaySpeedClass (velocidade classificativa de construção de jogadas da equipa)
    - buildUpPlayDribbling (capacidade de drible da equipa)
    - buildUpPlayDribblingClass (capacidade classificativa de drible da equipa)
    - buildUpPlayPassing (efetividade dos passes da equipa)
    - buildUpPlayPassingClass (efetividade classificativa dos passes da equipa)

O objetivo do nosso trabalho passa por estimar a diferença de golos para cada partida, calculado da seguinte forma (a diferença pode ser negativa significando vitória da equipa de fora):

$ dif_{golos}=golos_{casa} - golos_{fora} $

A base de dados possui um extenso set de informação, no entanto, nem todo o conteúdo é relevante para a nossa análise. Para tal, passaremos à explicação das técnicas e métodos utilizados para a seleção de dados.

Abordagem e Pré-Processamento de Dados

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
# Read all files
country = pd.read_csv('Country_Original.csv')
league = pd.read_csv('League_Original.csv')
match = pd.read_csv('Match_Original.csv')
player = pd.read_csv('Player_Original.csv')
```

```
player_attributes = pd.read_csv('Player_Attributes_Original.csv')
team = pd.read_csv('Team_Original.csv')
team_attributes = pd.read_csv('Team_Attributes_Original.csv')

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

Organização e Limpeza de Dados

Pretendemos nesta secção do trabalho reduzir e tratar os dados de forma a que não exista informação irrelevante para o modelo que pretendemos construir.

- Eliminação de Valores Nulos

De forma a termos uma base de dados organizada e sem ruído, decidimos realizar várias alterações e tratamentos à base de dados fornecida. Começamos por remover os atributos que consideramos que não acrescentariam valor ao nosso algoritmo.

In [3]:

```
# Eliminar atributos relativos à posição dos jogadores
player_attributes.drop(player_attributes.columns.to_series()["potential":], axis=1, inplace = True
)
```

Neste caso escolhemos **remover os valores duplicados**, caso estes existam

In [4]:

```
# Eliminar duplicados caso existam
player_attributes.drop_duplicates(keep = False, inplace = True)
```

Dado que consideramos o atributo *overall_rating* importante na avaliação do jogador, caso este atributo seja nulo, deverá ser removido pois sem essa informação esse jogador não seria relevante.

In [5]:

```
# Verficar se existe overall_ratings null
player_attributes.dropna(axis=0, how="any", subset =['overall_rating'], inplace=True)
player_attributes['overall_rating'].isnull().sum()
```

Out[5]:

```
0
```

De forma a organizar e limpar os dados, decidimos, uma vez mais, remover colunas que na nossa opinião não seriam fulcrais para o desenvolvimento deste projeto, dando assim preferência a um conjunto de dados bem organizado.

In [6]:

```
# Eliminar atributos relativos à posição dos jogadores
match.drop(match.columns.to_series()["home_player_X1":"away_player_Y11"], axis=1, inplace = True)
# Eliminar os seguintes atributos -> shoton, shotoff, foulcommit, card, cross, corner, possession
match.drop(match.columns.to_series()["goal":"possession"], axis=1,inplace = True)
```

In [7]:

```
#Organizar dados por datas, do passado para o presente
match['date'] = match['date'].astype('datetime64[ns]')
match = match.sort_values(by=['date'], ascending=True)
```

Todas as partidas de futebol registadas que **não tiverem nenhuma informação sobre jogadores** são consideradas *lixo ou ruido* e consequentemente são removidas

In [8]:

```
#Eliminar todos as linhas que não tenham nenhuma info sobre os jogadores
```

```
match = match.dropna(axis=0, how="all", subset=match.columns.to_series()["home_player_1":"away_player_11"])
```

In [9]:

```
#Eliminar todas as linhas que não tenham nenhuma odd relativa ao jogo
match.dropna(axis=0, how="all", subset=match.columns.to_series()["B365H":], inplace=True)
```

Foi definido um limite mínimo de **8 Jogadores**, isto é, caso alguma linha da tabela *match* contenha informação sobre menos de 8 jogadores, esta será imediatamente removida.

In [10]:

```
#Eliminar partidas em que não são conhecidos pelo menos 8 jogadores que jogaram (para a Equipa da Casa)
match = match.dropna(axis=0, thresh=8, subset=match.columns.to_series()["home_player_1":"home_player_11"])
#Verificação
match.loc[match.loc[:,'home_player_1':'home_player_11'].count(axis=1) < 8]
```

Out[10]:

id	country_id	league_id	season	stage	date	match_api_id	home_team_api_id	away_team_api_id	home_team_goal	away_team_gc

In [11]:

```
#Eliminar partidas em que não são conhecidos pelo menos 8 jogadores que jogaram (para a Equipa de Fora)
match = match.dropna(axis=0, thresh=8, subset=match.columns.to_series()["away_player_1":"away_player_11"])
#Verificação
match.loc[match.loc[:,'away_player_1':'away_player_11'].count(axis=1) < 8]
```

Out[11]:

id	country_id	league_id	season	stage	date	match_api_id	home_team_api_id	away_team_api_id	home_team_goal	away_team_gc

In [ ]:

```

```

- Substituição de Valores Nulos

Incrementalmente fomos realizando **merges** de tabelas, com o objetivo de juntar toda a informação de todas as tabelas, de uma forma organizada, em apenas um data set.

Desta forma, quando analisamos um *match* somos capazes de ver os atributos dos jogadores envolvidos neste jogo, em vez de apenas vermos os seus id's. Após efetuar estes merges, tratamos de remover ou, como nem todos os valores nulos necessitam de ser removidos, estimar alguns destes valores (nomeadamente no que toca a dados relativos à equipa, em que a falta de informação de um ou dois jogadores não teria um impacto grande sobre os mesmos).

In [12]:

```
#Passar o formato dos atributos 'date' para formato de tempo
match['date'] = match['date'].astype('datetime64[ns]')
player_attributes['date'] = player_attributes['date'].astype('datetime64[ns]')
team_attributes['date'] = team_attributes['date'].astype('datetime64[ns]')
```

In [13]:

```
#Create copy of the match and player dataframe
match_copy = match.copy()
player_copy = player.copy()
player_height = player_copy[["player_api_id", "height","weight"]].copy()
```

Merge para fazer com o que nosso data_set passe a ter informações sobre os jogadores, utilizando os id's destes. Neste caso, estando-se a tratar da altura e do peso de cada jogador, fomos individualmente buscar o atributo do jogador para no fim serem criadas duas novas colunas para cada equipa *Média do peso*, *Média da altura*

In [14]:

```
# Merge das alturas e pesos para casa jogador em específico - Casa
for i in range(1,12):
    match_copy = match_copy.merge(player_height, how='left', left_on=['home_player_'+'%s'%i], right
_on=['player_api_id'], suffixes=('', '_home_'+'%s'%i), validate="m:1")
```

In [15]:

```
# Eliminar as colunas do player_api_id e o mudar o nome de height para height_home_1
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='player_api_id')))]
match_copy.rename(columns={"height": "height_home_1"}, inplace=True)
match_copy.rename(columns={"weight": "weight_home_1"}, inplace=True)
```

In [16]:

```
# Fazer média das altura e pesos
match_copy['mean_height_home']= match_copy[list(match_copy.filter(regex='height_home'))].mean(axis
=1)
match_copy['mean_weight_home']= match_copy[list(match_copy.filter(regex='weight_home'))].mean(axis
=1)
```

In [17]:

```
# Merge das alturas e pesos para casa jogador em específico - Fora
for i in range(1,12):
    match_copy = match_copy.merge(player_height, how='left', left_on=['away_player_'+'%s'%i], right
_on=['player_api_id'], suffixes=('', '_away_'+'%s'%i), validate="m:1")
```

In [18]:

```
# Eliminar as colunas do player_api_id e o mudar o nome de height para height_away_1
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='player_api_id')))]
match_copy.rename(columns={"height": "height_away_1"}, inplace=True)
match_copy.rename(columns={"weight": "weight_away_1"}, inplace=True)
```

In [19]:

```
# Fazer média das altura e pesos
match_copy['mean_height_away']= match_copy[list(match_copy.filter(regex='height_away'))].mean(axis
=1)
match_copy['mean_weight_away']= match_copy[list(match_copy.filter(regex='weight_away'))].mean(axis
=1)
```

Uma vez criadas as novas colunas, prosseguimos para a eliminação das colunas que apenas serviram de auxílio para calcular os valores utilizados nestas mesmas colunas novas.

In [20]:

```
# Eliminar colunas com alturas e pesos dos jogadores
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='height_home_')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='height_away_')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='weight_home_')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='weight_away_')))]
```

No que diz respeito às **odds** disponíveis para cada jogo, concluimos que a melhor abordagem possível seria realizar uma média das odds disponíveis para todas as possibilidades. Isto é, realizamos uma média das odds que todos os sites disponibilizaram para a possibilidade da **equipa da casa ganhar, para o empate e para a equipa de fora ganhar** , ficando assim com apenas 3 colunas e permitindo também trocar os valores nulos existentes pela média calculada.

In [21]:

```
#Criar Listas com os labbels das odds para Home, Draw, Away
home_ods = [a for a in match_copy.columns if a.endswith('H')]
draw_ods = [b for b in match_copy.columns if b.endswith('D')]
away_ods = [c for c in match_copy.columns if c.endswith('A')]
```

In [22]:

```
# Fazer média das odds
match_copy['mean_ods_home'] = match_copy[home_ods].mean(axis=1)
match_copy['mean_ods_draw'] = match_copy[draw_ods].mean(axis=1)
match_copy['mean_ods_away'] = match_copy[away_ods].mean(axis=1)
```

In [23]:

```
#Retirar colunas das ods
odds_indices_drop = ['B365H','B365D','B365A','BWH','BWD','BWA','IWH','IWD','IWA','LBH','LBD','LBA',
'PSH','PSD','PSA','WHH','WHD','WHA','SJH','SJD','SJA','VCH','VCD','VCA','GBH','GBD','GBA','BSH','BS
D','BSA']
match_copy.drop(columns=odds_indices_drop, inplace=True)
```

Um dos atributos essenciais é também o **overall_rating** de cada jogador.

Para cada jogador alinhado na equipa fomos pesquisar o seu *overall_rating*. É de notar que esta pesquisa foi baseada na data mais recente da sua atualização, visto um jogador possui diversos registos, em datas diferentes, de acordo com a sua evolução ao longo do tempo e foi necessário ter o cuidado de selecionar o registo mais recente de acordo com a data da partida.

Fizemos então merge com o nosso dataset, e demos lugar à criação de colunas auxiliares que continham os valores destes **overall_ratings**.

Depois de calculada a media do *overall_rating* da equipa, criamos então a coluna **mean_or_home** e **mean_or_away** (onde "*or*" é abreviação para *overall rating*) e procedemos para a eliminação das colunas auxiliares utilizadas.

In [24]:

```
# Ir buscar os overall_ratings para cada jogador a jogar a partida - Casa
player_ratings = player_attributes[["player_api_id", "date","overall_rating"]].copy()
for i in range(1,12):
    match_copy = match_copy.merge(player_ratings, how='outer', left_on=['home_player_'+'%s'%i], rig
ht_on=['player_api_id'], suffixes=('', '_home'), validate="m:m")
    index_to_drop1 = match_copy[ (match_copy['date_home'] > match_copy['date'])].index
    match_copy.drop(index_to_drop1 , inplace=True)
    match_copy.drop_duplicates(subset= ['id'], keep='first', inplace=True)
    match_copy.rename(columns={"overall_rating": "or_home_player_"+'%s'%i}, inplace=True)
    match_copy.drop(columns=['player_api_id','date_home'] , inplace=True)
```

In [25]:

```
# Ir buscar os overall_ratings para cada jogador a jogar a partida - Fora
for i in range(1,12):
    match_copy = match_copy.merge(player_ratings, how='outer', left_on=['away_player_'+'%s'%i], rig
ht_on=['player_api_id'], suffixes=('', '_away'), validate="m:m")
    index_to_drop1 = match_copy[ (match_copy['date_away'] > match_copy['date'])].index
    match_copy.drop(index_to_drop1 , inplace=True)
    match_copy.drop_duplicates(subset= ['id'], keep='first', inplace=True)
    match_copy.rename(columns={"overall_rating": "or_away_player_"+'%s'%i}, inplace=True)
    match_copy.drop(columns=['player_api_id','date_away'] , inplace=True)
```

In [26]:

```
# Criar média dos Overall Ratings da equipa da casa e fora, assim como eliminar os valores
associados aos jogador dado que
# agora só nos interessa a média

match_copy['mean_or_home']= match_copy[list(match_copy.filter(regex='or_home_player'))].mean(axis=1
)
match_copy['mean_or_away']= match_copy[list(match_copy.filter(regex='or_away_player'))].mean(axis=1
)
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='or_away_player_')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='or_home_player_')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='home_player')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='away_player')))]
```

In [27]:

```
#Eliminar valores adicionados ao match devido ao outter join e que não fazem sentido
match_copy = match_copy.dropna(axis=0, subset=['id'])
```

No que diz respeito à analise do resultado, decidimos remover as duas colunas com os golos de cada equipa e criar uma apenas **dif_goals** que representa a diferença de golos:

- > 0 Equipa da casa ganha
- = 0 Empate
- < 0 Equipa de fora ganha

In [28]:

```
# Cálculo da diferença de golos
match_copy["dif_goals"] = match_copy.home_team_goal - match_copy.away_team_goal
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='home_team_goal')))]
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='away_team_goal')))]
```

In [29]:

```
match_copy = match_copy[match_copy.columns.drop(list(match_copy.filter(regex='country_id')))]
```

In [30]:

```
team_attributes_copy = team_attributes.copy()
```

In [31]:

```
colToDrop = [d for d in team_attributes_copy.columns if d.endswith('Class')]
team_attributes_copy.drop(columns=colToDrop, inplace=True)
```

Nesta classe *Team Attributes*, decidimos agrupar os valores disponíveis de *buildUp*, *chance..* e *defence..* em 3 propriedades : *offense_team_rates*, *center_team_rates*, *defense_team_rates*.

Esta geração foi feita utilizando a média dos valores anteriormente referidos e, caso dois ou mais forem valores inválidos, esta linha de dados será então removida pois, uma vez mais, consideramos que os dados não são suficientes para definir uma equipa.

In [32]:

```
# Verificação de que existem pelo menos dois parametros para avaliar a equipa
team_attributes_copy = team_attributes_copy.dropna(axis=0, thresh=2, subset=["buildUpPlaySpeed", "b
uildUpPlayDribbling" ,"buildUpPlayPassing"])
team_attributes_copy = team_attributes_copy.dropna(axis=0, thresh=2,
subset=["chanceCreationPassing","chanceCreationCrossing","chanceCreationShooting"])
team_attributes_copy = team_attributes_copy.dropna(axis=0, thresh=2, subset=["defencePressure","def
enceAggression","defenceTeamWidth"])
```

In [33]:

```
#Divide os atributos das equipas em 3 categorias : Atacante, medio e defesa
team_attributes_copy["offense_team_rates"] = team_attributes_copy[list(team_attributes_copy.filter
(regex='buildUp'))].mean(axis=1)
team_attributes_copy["center_team_rates"] = team_attributes_copy[list(team_attributes_copy.filter(
regex='chanceCreation'))].mean(axis=1)
team_attributes_copy["defense_team_rates"] = team_attributes_copy[list(team_attributes_copy.filter
(regex='defence'))].mean(axis=1)
```

In [34]:

```
#Elimina as colunas que não vão ser utilizadas. Uma vez que as colunas novas já se encontram na ta
bela
team_attributes_copy =
team_attributes_copy[team_attributes_copy.columns.drop(list(team_attributes_copy.filter(regex='bui
ldUp')))]
```

```
team_attributes_copy =
team_attributes_copy[team_attributes_copy.columns.drop(list(team_attributes_copy.filter(regex='chanceCreation')))]
team_attributes_copy =
team_attributes_copy[team_attributes_copy.columns.drop(list(team_attributes_copy.filter(regex='defence')))]
```

In [35]:

```
#Definir as rates a inserir na tabela principal
team_ratings = team_attributes_copy[["team_api_id" ,  "date", "offense_team_rates" , "center_team_rates", "defense_team_rates"]]
```

In [36]:

```
match_copy.isnull().sum()
```

Out[36]:

```
id                   0
league_id            0
season               0
stage                0
date                 0
match_api_id         0
home_team_api_id     0
away_team_api_id     0
mean_height_home     0
mean_weight_home     0
mean_height_away     0
mean_weight_away     0
mean_ods_home        0
mean_ods_draw        0
mean_ods_away        0
mean_or_home         0
mean_or_away         0
dif_goals            0
dtype: int64
```

In [37]:

```
#Merge para a tabela principal dos ratings da equipa da casa
match_copy = match_copy.merge(team_ratings, how='outer', left_on=["home_team_api_id"], right_on=['team_api_id'], suffixes=('', '_home'), validate="m:m")
```

In [38]:

```
#Tratar o merge da equipa da casa
index_to_drop1 = match_copy[ (match_copy['date_home'] > match_copy['date'])].index
match_copy.drop(index_to_drop1 , inplace=True)
match_copy.drop_duplicates(subset= ['id'], keep='last', inplace=True)
match_copy.rename(columns={"offense_team_rates": "offense_team_rates_home", "center_team_rates" : "center_team_rates_home", "defense_team_rates":"defense_team_rates_home"}, inplace=True)
match_copy.drop(columns=['date_home', 'team_api_id'] , inplace=True)
```

In [39]:

```
#Eliminar valores adicionados ao match devido ao outter join e que não fazem sentido
match_copy = match_copy.dropna(axis=0, subset=['id'])
```

In [40]:

```
match_copy = match_copy.merge(team_ratings, how='outer', left_on=["away_team_api_id"], right_on=['team_api_id'], suffixes=('', '_away'), validate="m:m")
```

In [41]:

```
#Tratar o merge da equipa de fora
index_to_drop1 = match_copy[ (match_copy['date_away'] > match_copy['date'])].index
```

```
match_copy.drop(index_to_drop , inplace=True)
match_copy.drop_duplicates(subset= ['id'], keep='last', inplace=True)
match_copy.rename(columns={"offense_team_rates": "offense_team_rates_away", "center_team_rates" : "
center_team_rates_away", "defense_team_rates":"defense_team_rates_away"}, inplace=True)
match_copy.drop(columns=['date_away', 'team_api_id'] , inplace=True)
```

In [42]:

```
#Eliminar valores adicionados ao match devido ao outter join e que não fazem sentido
match_copy = match_copy.dropna(axis=0, subset=['id'])
```

In [43]:

```
#match_copy.isnull().sum()
# Se descomentarmos o comando em cima podemos ver que existem dados relativos às equipas. Isto dev
e-se ao facto de o
# id destas não estar presenta na base de dados dos Team Attributes. Sendo assim podemos eliminar
os valores nulos
match_copy.dropna(inplace=True)
```

In [44]:

```
match_copy.shape
```

Out[44]:

```
(16918, 24)
```

In [45]:

```
match_copy = match_copy.round(2)
```

In [46]:

```
match_copy.to_csv('pre_processed_df', index=False)
```

O conjunto de dados final, pronto a ser entregue ao algoritmo é o seguinte:

In [47]:

```
match_copy.head()
```

Out[47]:

	id	league_id	season	stage	date	match_api_id	home_team_api_id	away_team_api_id	mean_height_home	mean_weight
0	20093.0	19694.0	2009/2010	35.0	2010-04-24	820489.0	8596.0	8429.0	179.83	
4	21281.0	19694.0	2014/2015	8.0	2014-09-27	1726064.0	9927.0	8429.0	179.65	
6	20158.0	19694.0	2010/2011	10.0	2010-10-30	840285.0	9927.0	8429.0	181.49	
9	20265.0	19694.0	2010/2011	27.0	2011-02-19	840386.0	9927.0	8429.0	181.26	
13	21202.0	19694.0	2014/2015	30.0	2015-03-20	1726195.0	9927.0	8429.0	179.88	

Modelos de Aprendizagem (Regressão)

Após o tratamento e pré-processamento de dados, é possivel aplicar os algoritmos de aprendizagem supervisionada sobre os mesmos com o objetivo final de prever a diferença de golos entre duas equipas a jogar uma partida.
Como algoritmos de aprendizagem escolhemos utilizar:

- Àrvores de decisão
- Redes Neuronais

- Máquinas de Suporte Vetorial

Inicialmente é necessário importar um conjunto de bibliotecas que permitem e facilitam o uso destes modelos e fazer a leitura do ficheiro construído no pré-processamento.

In [48]:

```
import matplotlib.pyplot as plt
import seaborn as sns

from time import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, QuantileTransformer
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR
from sklearn.metrics import explained_variance_score, max_error, mean_squared_error, r2_score,
mean_absolute_error
```

In [49]:

```
# ler ficheiro pré-processado
df = pd.read_csv('pre_processed_df')
```

Após a leitura do ficheiro é necessário dividir os atributos de entrada e saída. Neste caso possuímos um atributo de saída (diferença de golos, valor a prever), portanto, atribuimos esse mesmo a 'y' e os restantes a 'X'.

In [50]:

```
#Divisão entre entradas e saída

X = df.drop(['id', 'league_id', 'season', 'stage', 'date','dif_goals'],axis=1)

y = df['dif_goals']
```

Tendo os dados divididos em entradas e saídas, o próximo passo trata a divisão dos dados num conjunto para treino e num conjunto para teste. Esta divisão é facilitada pela função de 'train_test_split()' disponibilizada pela biblioteca 'scikit-learn'.

O conjunto de treino é responsável pela construção do modelo de aprendizagem. É através deste que o modelo que, iterativamente, são ajustados os seus parâmetros de forma a otimizar o resultado produzido.
O conjunto de teste tem a função de, como o nome indica, avaliar o desempenho do modelo produzido pelo conjunto de treino e verificar se os resultados produzidos coincidem com o resultado verdadeiro.

Dada a importância do treino, optamos por utilizar 85% para o mesmo e os restantes 15% para teste.

É importante ter em conta que os dados apresentam grandezas e escalas diferentes sendo importante a sua normalização de forma a garantir integridade dos mesmos.

In [51]:

```
# Dividir dados em sets de Treino e Teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.15)

# Normalização de Dados
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- Árvore de Decisão

In [52]:

```
# Treinar o modelo - Árvores de Decisão
```

```
tick = time()

rfr = RandomForestRegressor(n_estimators=350)
rfr.fit(X_train, y_train)
prediction_rfr = rfr.predict(X_test)

print("Done in {:.3f}s".format(time() - tick))
```

```
Done in 58.994s
```

In [53]:

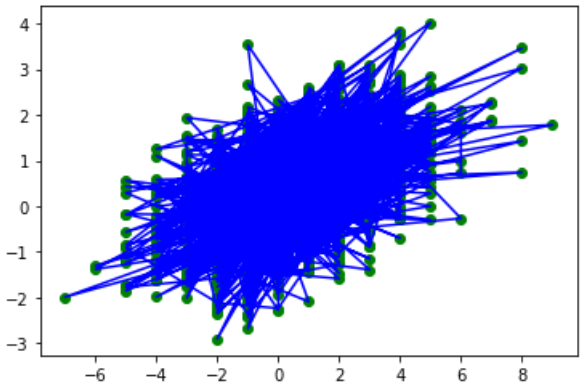
```
# Imprimir métricas para avaliação do desempenho do modelo

print("Mean Absolute Error:",mean_absolute_error(y_test, prediction_rfr))
print("Mean Square Error:",mean_squared_error(y_test, prediction_rfr))
print("Max Error:",max_error(y_test, prediction_rfr))
print("R2 Score:",r2_score(y_test, prediction_rfr))
```

```
Mean Absolute Error: 1.2786569852527299
Mean Square Error: 2.7198328331805537
Max Error: 7.257142857142857
R2 Score: 0.20743424822243095
```

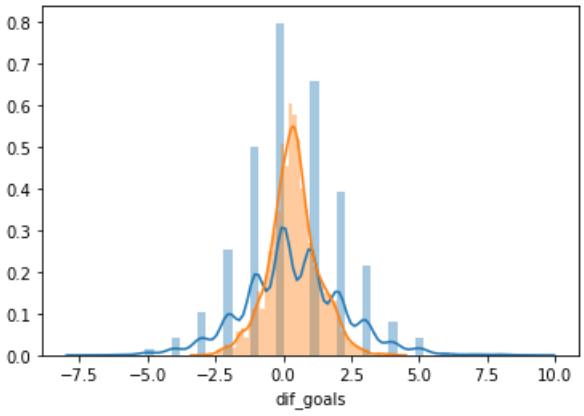
In [54]:

```
plt.scatter(y_test, prediction_rfr,color='g')
plt.plot(y_test, prediction_rfr,color='b')
plt.show()
```



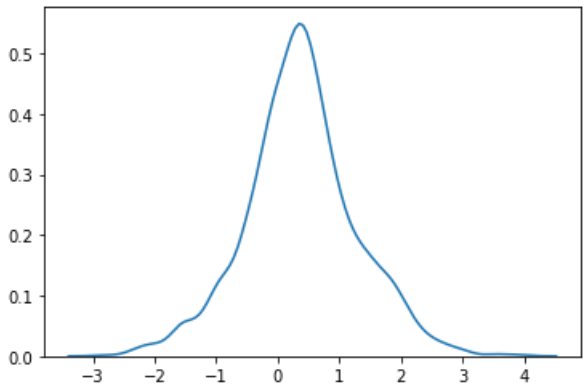
In [55]:

```
sns.distplot(y_test)
sns.distplot(prediction_rfr);
```



In [56]:

```
sns.kdeplot(prediction_rfr);
```



- Rede Neuronal

In [57]:

```
# Treinar o modelo - Rede Neuronal
tick = time()

mlp = make_pipeline(QuantileTransformer(),
MLPRegressor(hidden_layer_sizes=(50,50),learning_rate_init=0.01,early_stopping=True))
mlp.fit(X_train, y_train)
prediction_mlp = mlp.predict(X_test)

print("Done in {:.3f}s".format(time() - tick))
```

```
Done in 2.039s
```

In [58]:

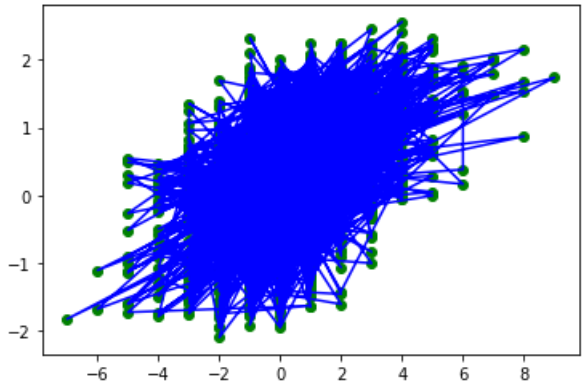
```
# Imprimir métricas para avaliação do desempenho do modelo

print("Mean Absolute Error:",mean_absolute_error(y_test, prediction_mlp))
print("Mean Square Error:",mean_squared_error(y_test, prediction_mlp))
print("Max Error:",max_error(y_test, prediction_mlp))
print("R2 Score:",r2_score(y_test, prediction_mlp))
```

```
Mean Absolute Error: 1.2708712040075723
Mean Square Error: 2.6745511612404465
Max Error: 7.259758235371413
R2 Score: 0.22062943504609644
```

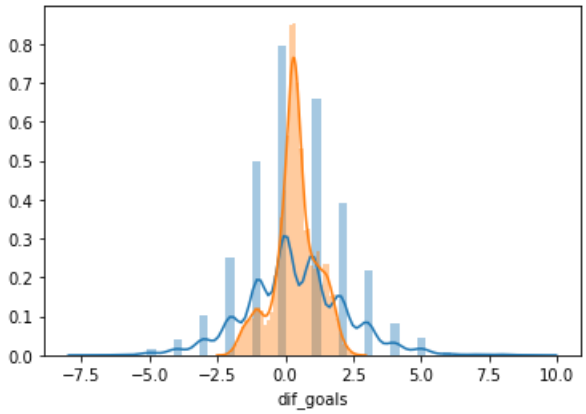
In [59]:

```
plt.scatter(y_test, prediction_mlp,color='g')
plt.plot(y_test, prediction_mlp,color='b')
plt.show()
```



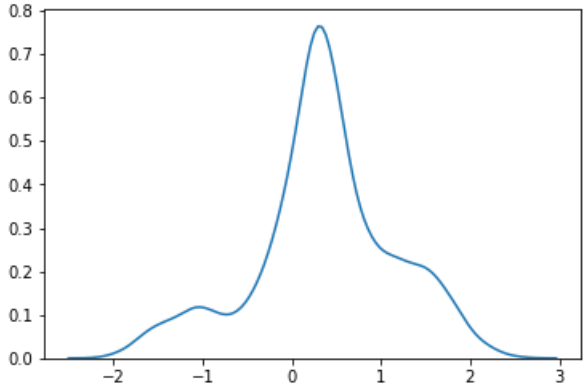
In [60]:

```
sns.distplot(y_test)
sns.distplot(prediction_mlp);
```



In [61]:

```
sns.kdeplot(prediction_mlp);
```



- Máquina de Suporte Vetorial

In [62]:

```
# Treinar o modelo - SVR
tick = time()

svr= SVR()
svr.fit(X_train, y_train)
prediction_svr = svr.predict(X_test)

print("Done in {:.3f}s".format(time() - tick))
```

```
Done in 12.064s
```

In [63]:

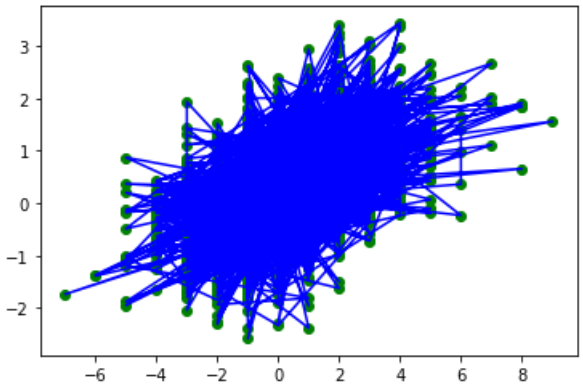
```
print("Mean Absolute Error:",mean_absolute_error(y_test, prediction_svr))
print("Mean Square Error:",mean_squared_error(y_test, prediction_svr))
print("Max Error:",max_error(y_test, prediction_svr))
print("R2 Score:",r2_score(y_test, prediction_svr))
```

```
Mean Absolute Error: 1.2875190294120864
Mean Square Error: 2.753207006523374
Max Error: 7.4403869537588285
R2 Score: 0.19770893479040097
```

In [64]:

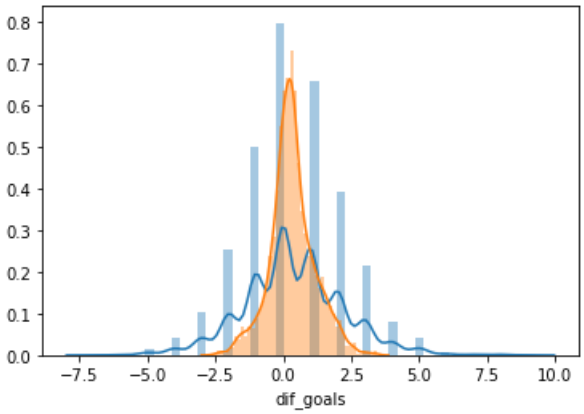
```
plt.scatter(y_test, prediction_svm, color='red')
```

```
plt.scatter(y_test, prediction_svr,color='g')
plt.plot(y_test, prediction_svr,color='b')
plt.show()
```



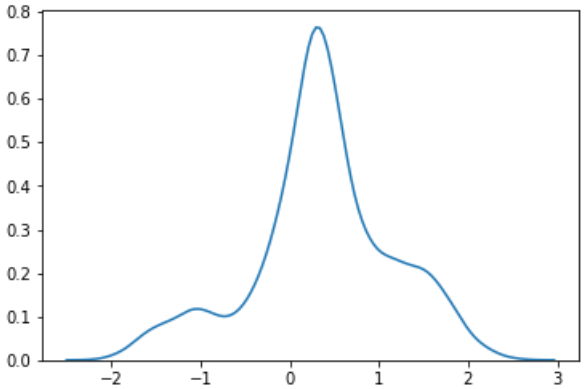
In [65]:

```
sns.distplot(y_test)
sns.distplot(prediction_svr);
```



In [66]:

```
sns.kdeplot(prediction_mlp);
```



Conclusão e Discussão de Resultados

Através de uma observação dos resultados obtidos em cima, concluímos que os algoritmos de aprendizagem produziram resultados semelhantes, com um ligeiro melhor desempenho do modelo da Rede Neuronal.

Para os modelos finais dos 3 algoritmos obtivemos valores à volta de 1.27/1,27/1,28 (respetivamente para Àrvore de Decisão, Rede Neuronal e Máquina de Suporte Vetorial) do Erro Absoluto Médio ("Mean Absolute Error") indicando que a previsão está geralmente a esse valor da realidade. Embora possa parecer bastante, o valor é justificado pela existência de um número elevado de *outliers* que acontece quando a diferença de golos de uma partida é alta, algo que é de difícil previsão. É possível verificar também a existência

destes *outliers* através do Erro Máximo registado de 7 golos.

Foram registados para o coeficiente R^{2} valores entre 0.19 e 0.23, que se revelam bons, dado que o assunto estudado é altamente dependente da ação humana, e dificilmente consegue ser prevista de forma precisa.

Com este trabalho foi possível aprofundar os nossos conhecimentos sobre Machine Learning, mais especificamente sobre Aprendizagem Supervisionada, mostrando-se assim o grupo satisfeito com o tema escolhido.

Foi possível visualizar a aplicabilidade prática dos algoritmos implementados, não só nos dados fornecidos para este trabalho, como também para problemas bastante atuais. Assim, o nosso nível de abstração em relação à área aumentou, tendo sido identificada outra regalia deste tema.

As grandes dificuldades encontradas incidiram principalmente no tratamento de dados pois estes continham muitos valores em falta e informação necessária espalhada por diversas tabelas da base de dados inicial. Este problema exigiu um esforço adicional por parte do grupo , de forma a compor o conjunto de dados pretendido para ser entregue aos modelos de aprendizagem.

Por último, concluímos que os modelos apresentaram resultados de acordo com os esperados e foi possível perceber a elevada importância que a aplicabilidade de Machine Learning tem para a actualidade.

Referências

API's e Documentação de SciKit Learn e Pandas:

- https://scikit-learn.org/stable/modules/classes.html
- https://pandas.pydata.org/docs/reference/index.html#api

Artigos e Projetos relevantes:

- https://www.researchgate.net/publication/257048220_Machine_Learning_for_Soccer_Analytics
- https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1718-ug-projects/Corentin-Herbinet-Using-Machine-Learning-techniques-to-predict-the-outcome-of-profressional-football-matches.pdf
- https://elitedatascience.com/python-machine-learning-tutorial-scikit-learn#step-1
- https://medium.com/data-hackers/implementando-regress%C3%A3o-linear-simples-em-python-91df53b920a8
- https://towardsdatascience.com/the-5-feature-selection-algorithms-every-data-scientist-need-to-know-3a6b566efd2
- https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html